ECE385

Fall 2021

Experiment 4

Introduction to SystemVerilog, FPGA, CAD, and 16-bit Adders

Ge Yuhao, Lou Haina
D231, Oct.17, 2021
TA: Chenhao Wang

1 Introduction

1.1 Summarize the high level function performed by the serial logic processor and the three adders

In this lab, we learn to extend the serial logic processor from 4-bit to 8-bit. With extended processor, we can do 8-bit operations(And,OR,...). Besides, we build Carry-ripple adder, Carry-Lookahead adder and Carry-Select adder, which can do add operation at one time. Comparing with Carry-ripple adder, Carry-Lookahead adder and Carry-select adder are faster but need more space and energy consumption

2 Serial Logic Processor

2.1 The top level schematic generated by the RTL viewer.

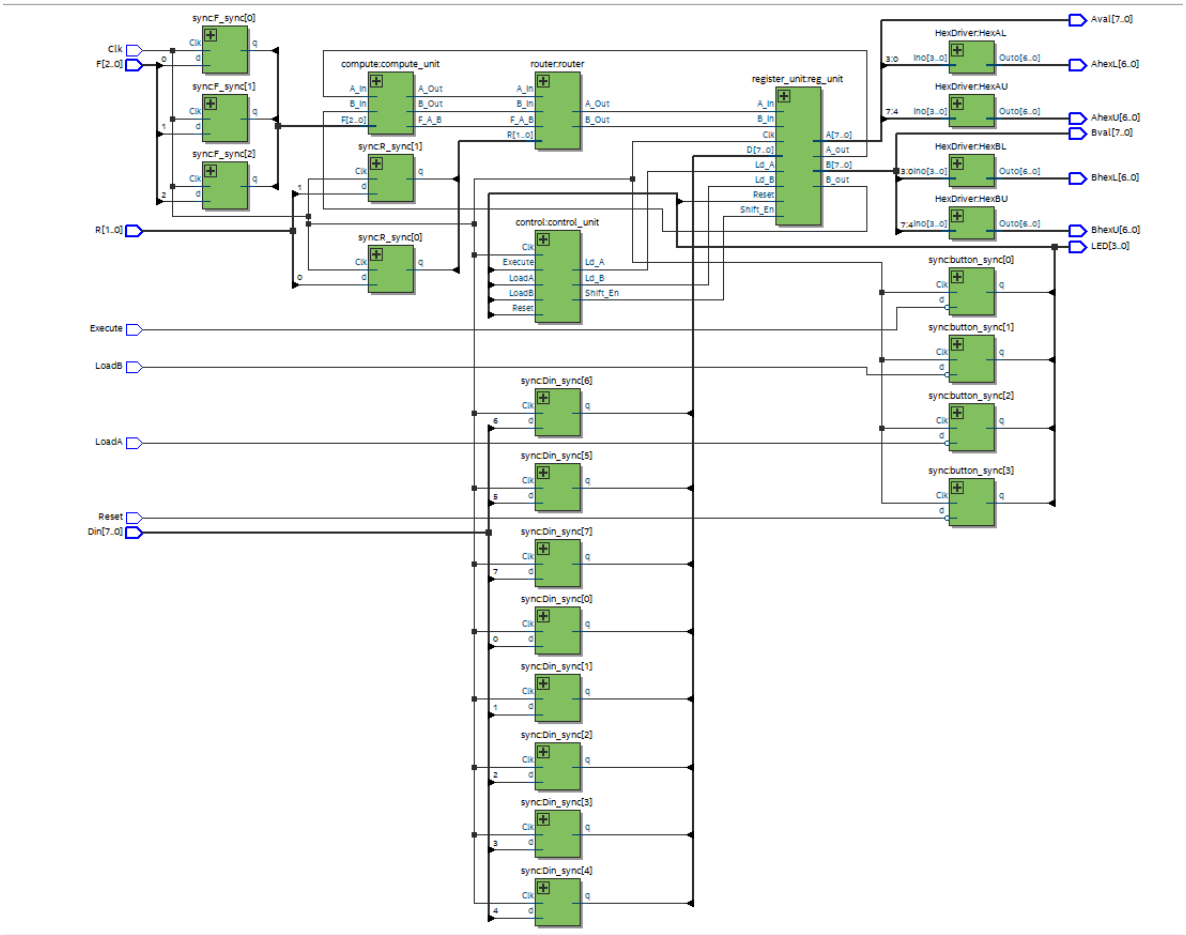
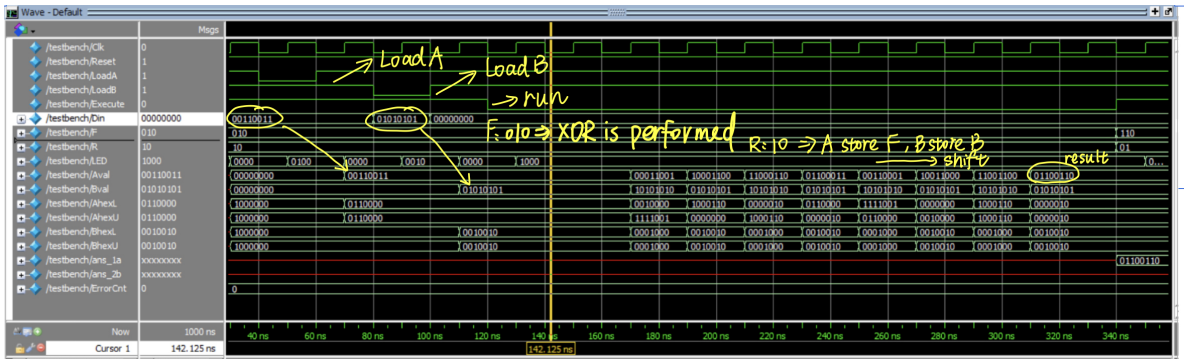


Figure 1: Toplevel schematic for Serial Logic Processor

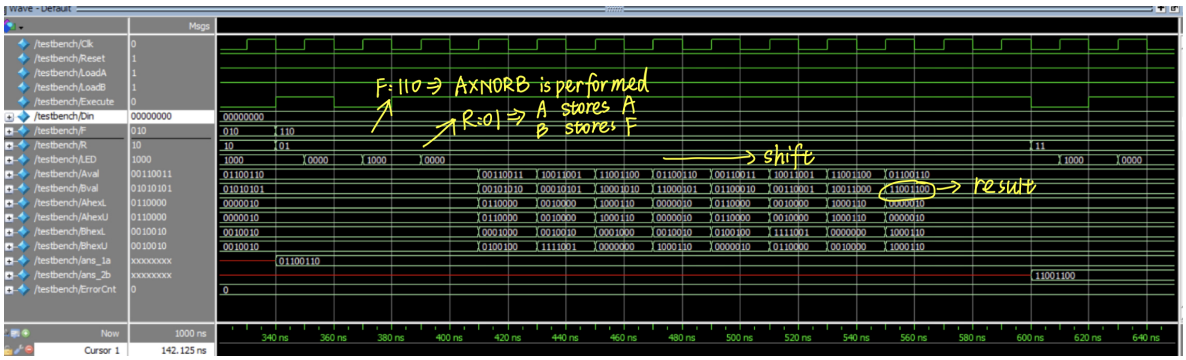
2.2 A short description about what was done with the provided code to extend it from 4 bits to 8 bits

Firstly, We change input and output bits number from 4 to 8. Then we modify the reg4 file to make the bits stored in the register is 8 bits. Also, in control file, we add four more states in Mealy machine state '0' to change clock cycle from 4 to 8 to adapt to 8 bits in the register.

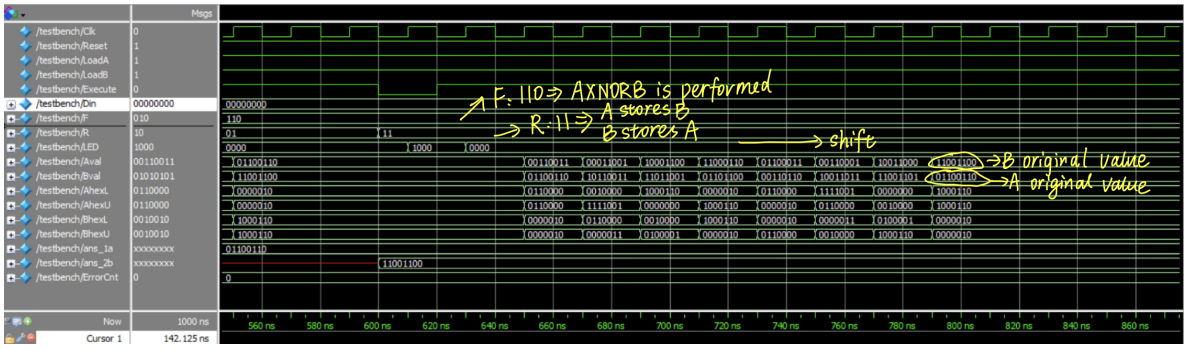
2.3 A simulation of the processor that has notes that give information



(a) test1



(b) test2



(c) test3

Figure 2: Simulation waveform for 8 bit logic processor

3 Adders

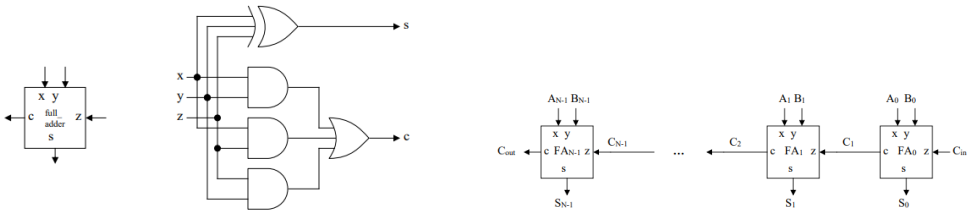
3.1 Ripple Carry Adder

3.1.1 Written description of the operation of your adder circuit

The Carry-Ripple Adder(CRA) can do 16-bit add operation with input A and B, and a carry-in (C_in) as inputs and output 16 bit sum (S) and a carry-out(C_out). CRA is constructed using 16 full-adders, and full-adders are linked together in series through the carry bits. When the binary inputs are provided, the full-adder of the least significant bit (LSB) will produce a sum (S0) and a carry-out (C1). The carry-out is fed to the carry-in of the second full-adder, which then produces a second sum

(S1) and a second carry-out (C2). The process ripples through all N bits of the adder, and settles when the full-adder of the most significant bit (MSB) outputs its sum (SN-1) and carry-out (Cout).

3.1.2 Block diagram



(a) Full-Adder Block diagram

(b) N-bit Carry-Ripple Adder Block Diagram

Figure 3: Carry-Ripple Adder block diagram

3.2 Carry Lookahead Adder

3.2.1 Written description of the architecture of the adder

1. Describe how the P and G logic are used
Instead of waiting on the actual carry-in values, Carry-Lookahead Adder (CLA) uses the concept of generating (G) and propagating (P) logic. CLA use immediate input A and B and predicts what its carry-out would be for any value of its carry-in, A carry-out is generated (G) if and only if both available inputs (A and B) are 1, regardless of the carry-in. The equation is : $G(A, B) = A \cdot B$. On the other hand, a carry-out has the possibility of being propagated (P) if either A or B is 1. The equation is : $G(A, B) = A \bigoplus B$. With P and G defined, the Boolean expression for the carry-out $C_i + 1$ giving a potential C_i is $C_{i+1} = C_i + P_i \cdot C_i$, C_{i+1} can only be determined by the first C_{in} and PG which is quickly determined by input A and B therefore the delays can be reduced.
2. Describe how you partitioned the adder
Instead of chain the 16 units adder together, to decrease the number of gates and avoid making CLA too large to be impractical to build, we first construct 4-bit CLAs and then use them to create a larger CLA in a hierarchical fashion. CLA was implemented in 4×4-bit instead of 16-bit. In the 4x4-bit hierarchical CLA design, the 16-bit inputs A and B are divided into groups of 4 bits. First, each group of 4 bits go through a 4-bit CLA. The P and G can be obtained by equation:

$$P_G = P_0 \cdot P_1 \cdot P_2 \cdot P_3$$

$$G_G = G_3 + G_2 \cdot P_3 + G_1 \cdot P_3 \cdot P_2 + G_0 \cdot P_3 \cdot P_2 \cdot P_1$$

We can derive Ps and Gs from 4 groups and use them to compute C_{out}, $C_{16} = G_{G12} + G_{G8} \cdot P_{G12} + G_{G4} \cdot P_{G8} \cdot P_{G12} + G_0 \cdot P_{G4} \cdot P_{G8} \cdot P_{G12} + C_0 \cdot P_{G0} \cdot P_{G4} \cdot P_{G8} \cdot P_{G12}$. In this way, we can build a upper level for 4-bit groups of CLAs.

3.2.2 Block diagram

1. Block diagram inside a single CLA (4-bits)

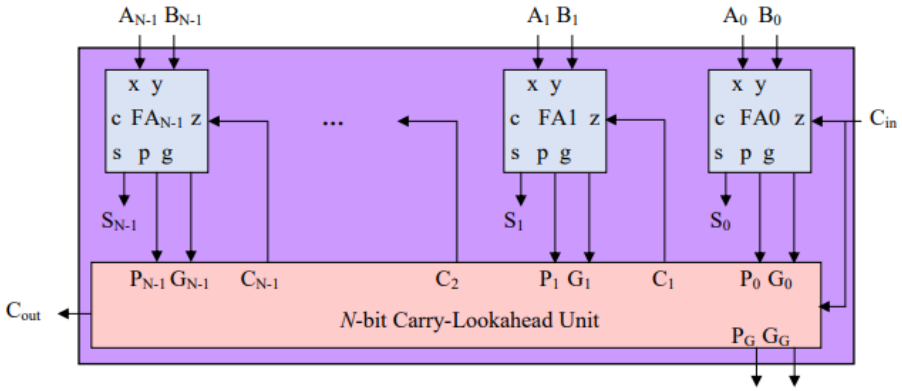


Figure 4: The block diagram of a N-bit Carry-Lookahead adder

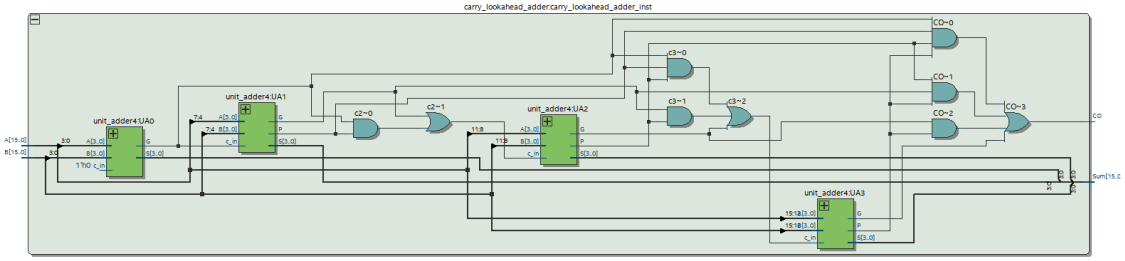


Figure 5: The RTL of a 4x4-bit Hierarchical Carry-Lookahead Adder

2. Block diagram of how each CLA was chained together

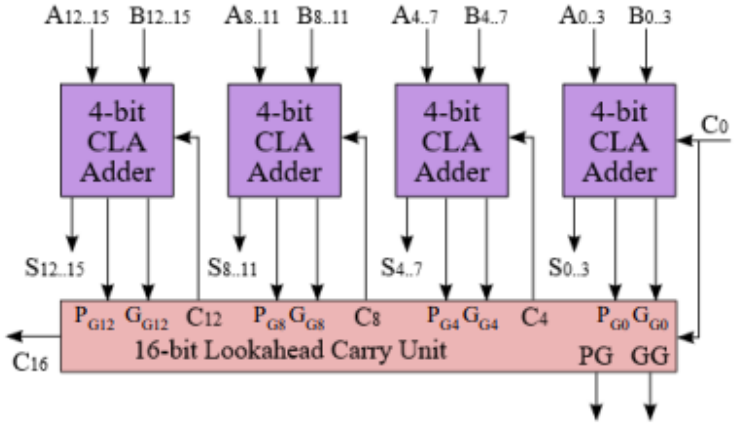


Figure 6: The block diagram of a 4x4-bit Hierarchical Carry-Lookahead Adder

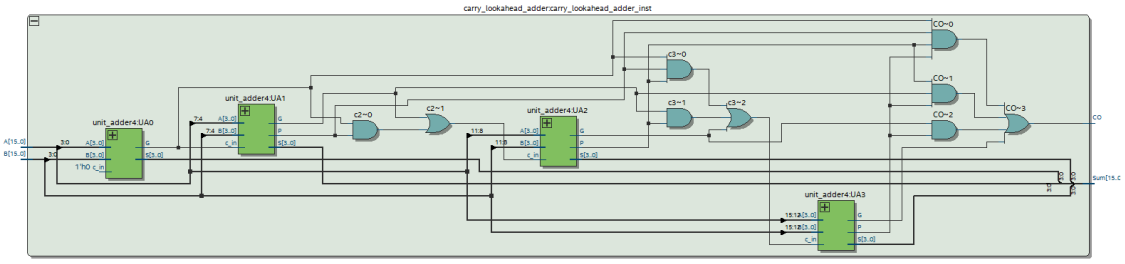


Figure 7: The RTL of a 4x4-bit Hierarchical Carry-Lookahead Adder

3.3 Carry Select Adder

3.3.1 Written description of the architecture of the adder

Carry-Select Adder (CSA) features another way to speed up the carry computation. It consists of two full adders (or CRAs if multiple bits are grouped) and a multiplexor. One adder computes the sum and carry-out based on the assumption that the carry-in is 0, and the other assumes that the carry-in is 1. In this way, both possible outcomes are pre-computed. Once the real carry-in arrives, the corresponding sum and carry-out is selected to be delivered to the next stage. By paying the price of almost twice the numbers of adders, we gain some speedup. Because we do adder operation by four groups of adders at the same time, we only need to wait for the selection delay.

3.3.2 Block Diagram of the whole CSA circuit containing adders, multiplexers, and glue logic

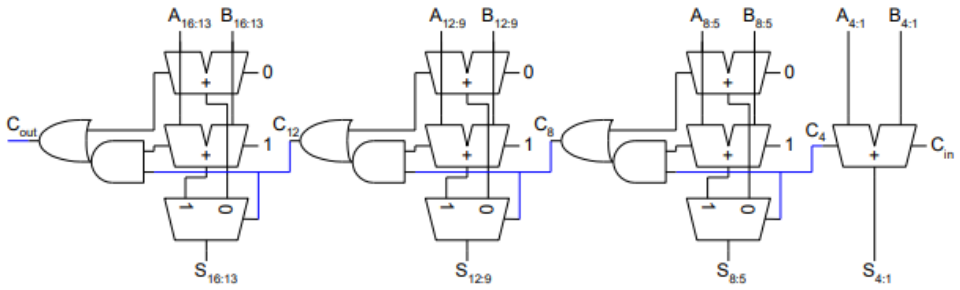


Figure 8: 16-bit Carry-Select Adder Block Diagram

3.4 Written purpose and operation of each module

Module:lab4_adders_toplevel.sv00
Inputs: Clk, LoadB, Reset, Run,[15:0] SW,
Outputs: [15:0] sum,co,[6:0] Ahex0, Ahex1, Ahex2, Ahex3, Bhex0, Bhex1, Bhex2, Bhex3
Description: This is a top level module which use module adder to compute sum and change it to hex to be shown on DE2 board.
Purpose: The module is used as a top level module to connect each module in this project, and receive input signal and compute it with adder, and change it to hex representation and show it on DE2 board.

Module: ripple_adder.sv
Inputs: [15:0]A,B
Outputs: [15:0] sum,co
Description: This is a carry-ripple adder with 4 small units of 4 full adder. It accepts A,B, carry in

bit and output its sum and carry out bit co.
Purpose: This module is used to adder A, B and carry bit.

Module: full_adder(in ripple_adder.sv)
Inputs: x,y,z
Outputs: s,c
Description: This is the smallest adder unit in ripple adder file
Purpose: This module accepts x,y and carry in bit z, and outputs sum s and carry out bit c.

Module: adder4(in ripple_adder.sv)
Inputs: c_in, [3:0]A,B,
Outputs: [3:0]S, c_out
Description: This is a middle layer adder made up of 4 full adders with 4 bit input and output.
Purpose: Add input A and B and output sum S and carry out bit.

Module: carry_lookahead_adder.sv
Inputs: [15:0]A,B
Outputs: [15:0] sum,co
Description: This is a carry-lookahead adder with 4 small units of unit adders (will be described below). It accepts A,B, carry in bit and output its sum and carry out bit co. Different from carry ripple adder, we need P and G of each unit adder to compute carry in and carry out bits between each unit adder.
Purpose: This module is used to adder A, B and carry bit.

Module: unit_adder (in carry_lookahead_adder.sv)
Inputs: A, B, C_in
Outputs: P, G, S
Description: This module is the smallest unit in carry-lookahead adder which also has compute value of P, G of this unit
Purpose: This module accepts x,y and carry in bit z, and outputs sum s, P ,G and carry out bit c.

Module: unit_adder4 (in carry_lookahead_adder.sv)
Inputs: c_in,[3:0]A,B
Outputs: P,G, [3:0]S,
Description: This is a middle level adder made up of 4 unit adders with 4 bit input and output.
Purpose: This module accepts A,B and carry in bit c_in, and outputs 4 bit sum s, also group value P ,G. Carry out value is not necessary.

Module: carry_select_adder.sv
Inputs: [15:0]A,B
Outputs: [15:0] sum,co
Description: This is a carry-lookahead adder with 3 small units of csa4 and one adder4 which is made up of 4 small units of full adder. It accepts A,B, carry in bit and output its sum and carry out bit co.
Purpose: This module is used as top level adder to add A, B and carry bit.

Module: csa4 (in carry_select_adder.sv)
Inputs: [3:0]A, [3:0]B, C_in
Outputs: [3:0]S, c_out
Description: This module is middle level unit for carry_select_adder which computes sum and carry out bit for 0 and 1 carry_in bit respectively. A 2-1 MUX is used to select the sum with carry in bit 0 or 1.
Purpose: This module accepts A,B and carry in bit c_in, and outputs sum S, and carry out bit c_out.

Module: MUX_4 (in carry_select_adder.sv)
Inputs: sel,[3:0]A,B
Outputs: [3:0]out,
Description: This is a 2-1 MUX which select 4 bit value A or B by input bit 0 or 1
Purpose: This module accepts carry_in bit from last group and select the correct sum with this carry_in bit.

3.5 Describe at a high level the area, complexity, and performance tradeoffs between the adders.

- For area comparation, CRA uses 114 LUTs, and CLA uses 126 LUTs, while CSA uses 123 LUTs, which means CLA and CSA use more gate and space.
- As for complexity, the CRA is the easiest to design, and the structure of the RTL is much mroe intuitive comparing with the other two.
- For performance, every simple adder unit in the CRA need to wait for the carry-in bit from the last adder, therefore it has 15 delays for 16 bit adder. For CLA, each unit can compute in parallel as the input of each unit doesn't depend on the former unit. The same for the CSA as it will give the result of both accepting "1" or "0" as former carry bit, and decide which output to use at last. The frequency of the CLA and CSA is much higher than the CRA, so the performance of the CLA and CSA is much higher.

3.6 Document the performance of each adder

	Carry-Ripple	Carry-Select	Carry-Lookahead
Memory	0	0	0
Frequency	62.1MHz	95.18MHz	82.08MHz
Total Power	155.8mV	161.80mV	161.94mV

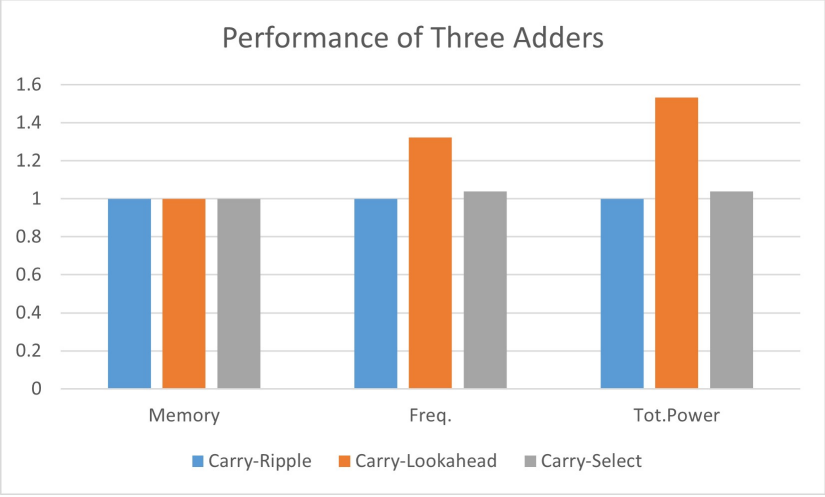


Figure 9: Performance of Three Adders

4 Answers to post-lab questions

4.1 Compare the usage of LUT, Memory, and Flip-Flop of your bit-serial logic processor exercise in the IQT with your TTL design in Lab 3.

- As for the usage of LUT, the TTL design has much less unit, we think this is because in SystemVerilog, each lookup table can store up to 4 inputs which is less than the chips in TTL.
- They both need 0 memory because all the message is stored in the register.
- The number of Flip-Flops in TTL design is less. We think this is because in SystemVerilog, it need more Flip-Flop to store bits which will than send to the lookup table while a TTL design needn't to do that.
- We think that the TTL design is better because it use much fewer LUT and Flip-Flop, which means the inner structure is much simpler. But it worse mention that we also look up the power usage of the two design and find that the power consumption of the TTL design is much higher, maybe it is because some chips will need more power even we only use part of its full function. So the TTl design may also have some drawbacks.

LUT	72
Memory	0
Flip-Flop	43
Total Power	158.6mW

Table 1: logic processor

LUT	10
Memory	0
Flip-Flop	10
Total Power	354.9mW

Table 2: TTL design

Figure 10: Design statics table for each adder

4.2 In the CSA for this lab, we asked you to creat a 4*4 hierarchy. Is this ideal? If not, how would you go about designing the ideal hierarchy on the FPGA

- We think it's very hard to decide which design is "ideal", as each design may have different merits. Some may gain a faster speed but need more space, while some may have simple structure but lower speed. But we try to give some hypothesis as below.
- First, the CSA we designed in this lab is not strictly a 4*4 hierarchy. We optimize the first unit by using only one 4-bit Carry-Ripple adder as the first carry bit must be an "0". This can reduce the total area and power consumption of our design.
- Second, the unit of our CSA is seven 4-bit Carry-Ripple adders, which can be changed to 4-bit Carry-Lookahead Adders which are faster. But considering the adder is only 4 bits, the speed of a CLA won't be much higher than CRA. Further more, the CLA will consume more energy and thus will produce more heat, which may influence the performance of the hardware. So whether to use this design should be considered more carefully by conducting more experiments to check the power consumption and maximun frequency of the design.

4.3 The statistics table for each adder and the explanation to the data

- As for the LUT, we can see that the LUT of CLA and CSA is more than CRA, which means that the CLA and CSA need more space. This does make sense as the structure of the CLA ans CSA is much more complex.
- As for the Frequency, we can see that the frequency of CLA and CSA is higher than CRA. This also meets our expectation as the procedure of CRA is linear, which means the latter units' input

depends on the former one, while the CLA and CSA don't have such limitation. The delay of CLA is 8 and the delay of CSA is 7 which are smaller than the delay of CRA which is 15, thus CLA and CSA have a larger frequency.

- As for the total Power, we can see that the power consumption of the CLA ans CSA is higher than the CRA. This does make sense. For the CLA, it needs more computation to decide each P and G to calculate each C, which raise power consumption. For the CSA, it compute two possible outputs for different carry bit input, which almost double the times of calculation. Therefore, the CLA and CSA must consume more power.

-

LUT	114
DSP	0
BRAM	0
Flip-Flop	105
Frequency	122MHz
Static Power	98.55mW
Dynamic Power	2.86mW
Total Power	155.94mW

Table 3: CRA

LUT	126
DSP	0
BRAM	0
Flip-Flop	105
Frequency	139.94MHz
Static Power	98.55mW
Dynamic Power	3.34mW
Total Power	156.30mW

Table 4: CLA

LUT	123
DSP	0
BRAM	0
Flip-Flop	105
Frequency	135.61MHz
Static Power	98.55mW
Dynamic Power	3.22mW
Total Power	156.16mW

Table 5: CSA

Figure 11: Design statics table for each adder

5 Conclusion

5.1 Describe any bugs and countermeasures taken during this lab.

- When we first redesign the 4-bit bit-serial logic processor to 8-bit, we failed to give the correct output. After debugging we found that we didn't add states to the Finite State Machine. Therefore, we add four more state into the FSM as the shifting time is increased from 4 to 8.

- When we design the CSA, we need to implement the 2-to-1 MUX and each input need to have 4 bits. Originally, we only build the MUX according to the IST which is only a 2-to-1 with each input having 1 bit, and we reuse this unit 4 times to realise 4-bits selection. Although this worked well which can give us correct ouput, but the code and the structure of our RTL design is not elegant and possibly not efficient. So we try to polish this but coming up with another MUX which integrated 4 bits as one input of the MUX.

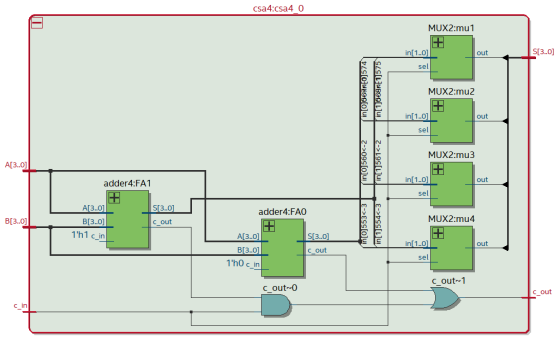


Figure 12: Original 2-to-1 MUX

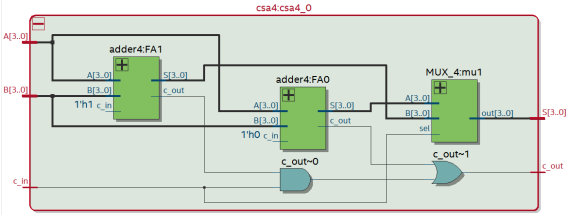


Figure 13: Modified 2-to-1 MUX

- When we design the 4x4-bit hierarchical CLA, we misunderstand the structure of each CLA unit. First, we forget to calculate the output S from each unit, and later we add the logic $S = A\hat{}B\hat{}C\_in$.

- When we cascade the four 4-bit CLAs by connecting the C_out from the previous 4-bit CLA to the C_in of the next 4-bit CLA, we find that we were trapped by the slow rippling of these carry bits again. To overcome this, we again learn the idea from the unit design to get each c_in by only Ps and Gs.

5.2 Was there anything ambiguous, incorrect, or unnecessarily difficult in the lab manual or given materials which can be improved for next semester? You can also specify what we did right so it doesn't get changed.

Generally this lab's instructions are very good which help us to understanding the used of SystemVerilog and Quartus quickly. The "Introduction to SystemVerilog" and "Introduction to Quartus Prime" PDF files are very good for rookies. However, there are still some ambiguous instrction. For example, when the code is ready on FPGA, we spend many time understanding the use of the switches and buttons because we don't know which button is *Reset* and which is *Run*. Besides, when we install the Driver of FGPA, we don't know where to find the Driver file in the Quartus Prime folder and update the Driver until we searched online.